oVirt Scheduler
Deep Dive

Agenda

oVirt

- Intro
- Modules & Code Samples
  - Filter
  - Weight Module
  - Load Balance
- Implementation & Flows
  - Engine
  - External Scheduling Proxy

Intro

The need

Re: [Users] How to define max number of running VMs on a host?

….

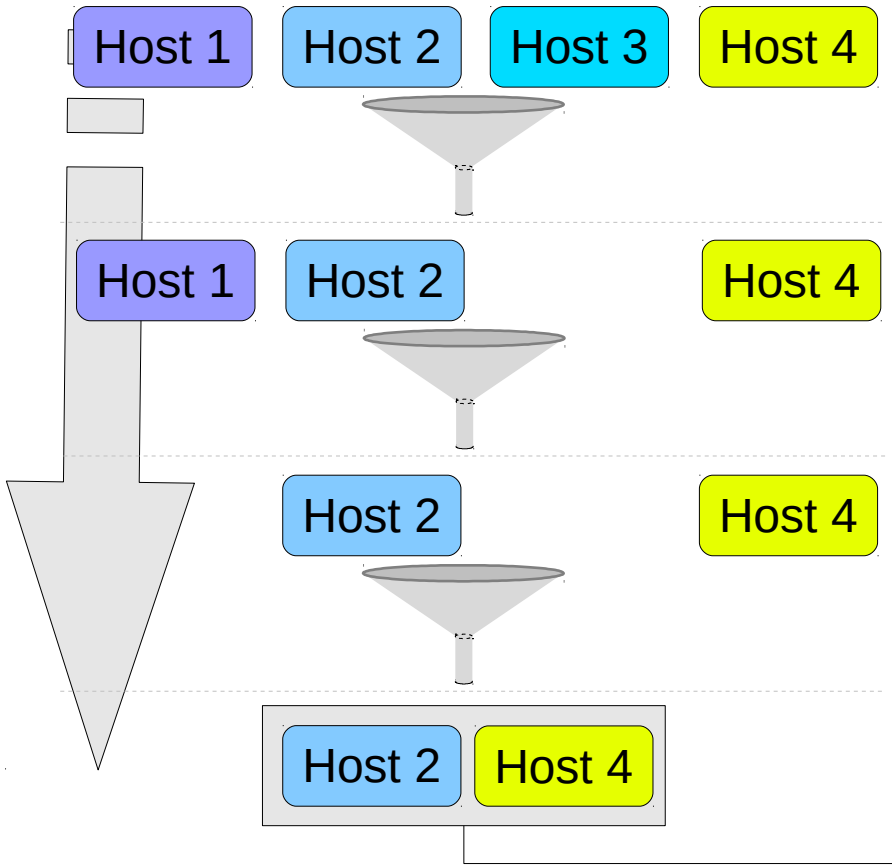
I have 4 graphic workstations with 3 graphic cards on each. I wanna passthrough graphic cards to the VMs one by one, since one workstation has only 3 cards, I must limit the number of running VM on a host to 3.

Intro

- Current oVirt Scheduler
  - Executes the selected distribution algorithm on the cluster:
    - Even Distribution
    - Power Saving
  - Selects a host to run/migrate VM on.
  - Balance: Selects a VM to migrate and Host to migrate to.
  - Only 2 distribution algorithms, taking into consideration only CPU usage
  - No way to construct a user defined scheduling policy

Intro

- The New Model*



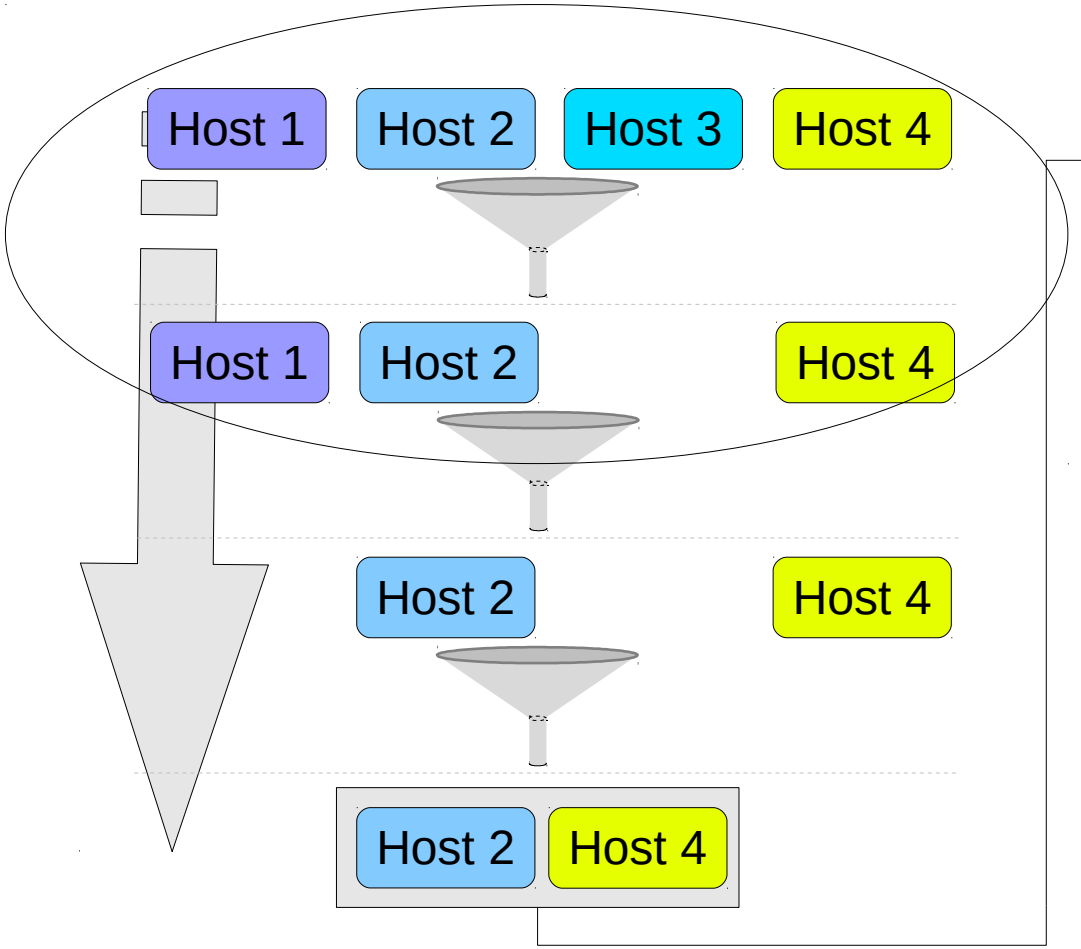
	func 1	func 2	sum
Factor	5	2	
Host 2	10	2	54
Host 4	3	12	**39***

***Host 4 sum: 3*5+12*2 = 39**

*Uses Nova Scheduling concepts.

Filter Module

- The New Model



	func 1	func 2	sum
Factor	5	2	
Host 2	10	2	54
Host 4	3	12	**39***

***Host 4 sum: 3*5+12*2 = 39**

Filter Module

oVirt

- A basic logic unit which filters out hypervisors who do not satisfy the hard constraints for placing a given VM
  - Clear cut logic
  - Easy to write and maintain
  - Chained up-dependently to allow complete filtering
  - Allows custom parameters
- Existing logic (pin-to-host, memory limitations, etc.) is translated into filters
- External filters written in python can be loaded into engine.

Let's go back to the example

Re: [Users] How to define max number of running VMs on a host?

….

I have 4 graphic workstations with 3 graphic cards on each. I wanna passthrough graphic cards to the VMs one by one, since one workstation has only 3 cards, I must limit the number of running VM on a host to 3.

Let's go back to the example

Re: [Users] How to define max number of running VMs on a host?

....

I have 4 graphic workstations with 3 graphic cards on each. I wanna passthrough graphic cards to the VMs one by one, since one workstation has only 3 cards, I must limit the number of running VM on a host to 3.

Filter: filters out hosts with number running of vms > 3

Filter Sample (python)

```python
class max_vms():
    '''returns only hosts with less running vms then the maximum'''

    #What are the values this module will accept, used to present
    #the user with options
    properties_validation = 'maximum_vm_count=[0-9]*'

    def do_filter(self, hosts_ids, vm_id, args_map):
        #open a connection to the rest api
        try:
            connection = API(url='http://host:port',
                             username='user@domain', password='')
        except BaseException as ex:
            #letting the external proxy know there was an error
            print >> sys.stderr, ex
            return

        #get our parameters from the map
        maximum_vm_count = int(args_map.get('maximum_vm_count', 100))

        #get all the hosts with the given ids
        engine_hosts = \
            connection.hosts.list(
                query=" or ".join(["id=%s" % u for u in hosts_ids]))

        #iterate over them and decide which to accept
        accepted_host_ids = []
        for engine_host in engine_hosts:
            if(engine_host and
                    engine_host.summary.active < maximum_vm_count):
                accepted_host_ids.append(engine_host.id)
        print accepted_host_ids
```

Filter Sample (python)

```
class max_vms():
    '''returns only hosts with less running vms then the maximum'''

    #What are the values this module will accept, used to present
    #the user with options
    properties_validation = 'maximum_vm_count=[0-9]*'

    def do_filter(self, hosts_ids, vm_id, args_map):
        #open a connection to the rest api
        try:
            connection = API(url='http://host:port',
                             username='user@domain', password='')
        except BaseException as ex:
            #letting the external proxy know there was an error
            print >> sys.stderr, ex
            return

        #get our parameters from the map
        maximum_vm_count = int(args_map.get('maximum_vm_count', 100))

        #get all the hosts with the given ids
        engine_hosts = \
            connection.hosts.list(
                query=" or ".join(["id=%s" % u for u in hosts_ids]))

        #iterate over them and decide which to accept
        accepted_host_ids = []
        for engine_host in engine_hosts:
            if(engine_host and
                    engine_host.summary.active < maximum_vm_count):
                accepted_host_ids.append(engine_host.id)
        print accepted_host_ids
```

Filter Sample (python)

```
class max_vms():
    '''returns only hosts with less running vms then the maximum'''

    #What are the values this module will accept, used to present
    #the user with options
    properties_validation = 'maximum_vm_count=[0-9]*'

    def do_filter(self, hosts_ids, vm_id, args_map):
        #open a connection to the rest api
        try:
            connection = API(url='http://host:port',
                             username='user@domain', password='')
        except BaseException as ex:
            #letting the external proxy know there was an error
            print >> sys.stderr, ex
            return

        #get our parameters from the map
        maximum_vm_count = int(args_map.get('maximum_vm_count', 100))

        #get all the hosts with the given ids
        engine_hosts = \
            connection.hosts.list(
                query=" or ".join(["id=%s" % u for u in hosts_ids]))

        #iterate over them and decide which to accept
        accepted_host_ids = []
        for engine_host in engine_hosts:
            if(engine_host and
                    engine_host.summary.active < maximum_vm_count):
                accepted_host_ids.append(engine_host.id)
        print accepted_host_ids
```

Filter Sample (python)

```
class max_vms():
    '''returns only hosts with less running vms then the maximum'''

    #What are the values this module will accept, used to present
    #the user with options
    properties_validation = 'maximum_vm_count=[0-9]*'

    def do_filter(self, hosts_ids, vm_id, args_map):
        #open a connection to the rest api
        try:
            connection = API(url='http://host:port',
                             username='user@domain', password='')
        except BaseException as ex:
            #letting the external proxy know there was an error
            print >> sys.stderr, ex
            return

        #get our parameters from the map
        maximum_vm_count = int(args_map.get('maximum_vm_count', 100))

        #get all the hosts with the given ids
        engine_hosts = \
            connection.hosts.list(
                query=" or ".join(["id=%s" % u for u in hosts_ids]))

        #iterate over them and decide which to accept
        accepted_host_ids = []
        for engine_host in engine_hosts:
            if(engine_host and
                    engine_host.summary.active < maximum_vm_count):
                accepted_host_ids.append(engine_host.id)
        print accepted_host_ids
```

Filter Sample (python)

```
class max_vms():
    '''returns only hosts with less running vms then the maximum'''

    #What are the values this module will accept, used to present
    #the user with options
    properties_validation = 'maximum_vm_count=[0-9]*'

    def do_filter(self, hosts_ids, vm_id, args_map):
        #open a connection to the rest api
        try:
            connection = API(url='http://host:port',
                             username='user@domain', password='')
        except BaseException as ex:
            #letting the external proxy know there was an error
            print >> sys.stderr, ex
            return

        #get our parameters from the map
        maximum_vm_count = int(args_map.get('maximum_vm_count', 100))

        #get all the hosts with the given ids
        engine_hosts = \
            connection.hosts.list(
                query=" or ".join(["id=%s" % u for u in hosts_ids]))

        #iterate over them and decide which to accept
        accepted_host_ids = []
        for engine_host in engine_hosts:
            if(engine_host and
                    engine_host.summary.active < maximum_vm_count):
                accepted_host_ids.append(engine_host.id)
        print accepted_host_ids
```

Filter Sample (Java)

```java
@Override
public List<VDS> filter(List<VDS> hosts,
        VM vm, Map<String,
        String> parameters,
        List<String> messages) {

    List<VDS> list = new ArrayList<VDS>();
    for (VDS vds : hosts) {
        Integer cores = SlaValidator.getInstance().getEffectiveCpuCores(vds);
        if (cores != null && vm.getNumOfCpus() > cores) {
            messages.add(VdcBllMessages.ACTION_TYPE_FAILED_VDS_VM_CPUS.toString());
            log.debugFormat("host {0} has less cores ({1}) than vm cores ({2})",
                    vds.getName(),
                    cores,
                    vm.getNumOfCpus());
            continue;
        }
        list.add(vds);
    }

    return list;
}
```

* getEffectiveCpuCores(): checks whether threads count as cores

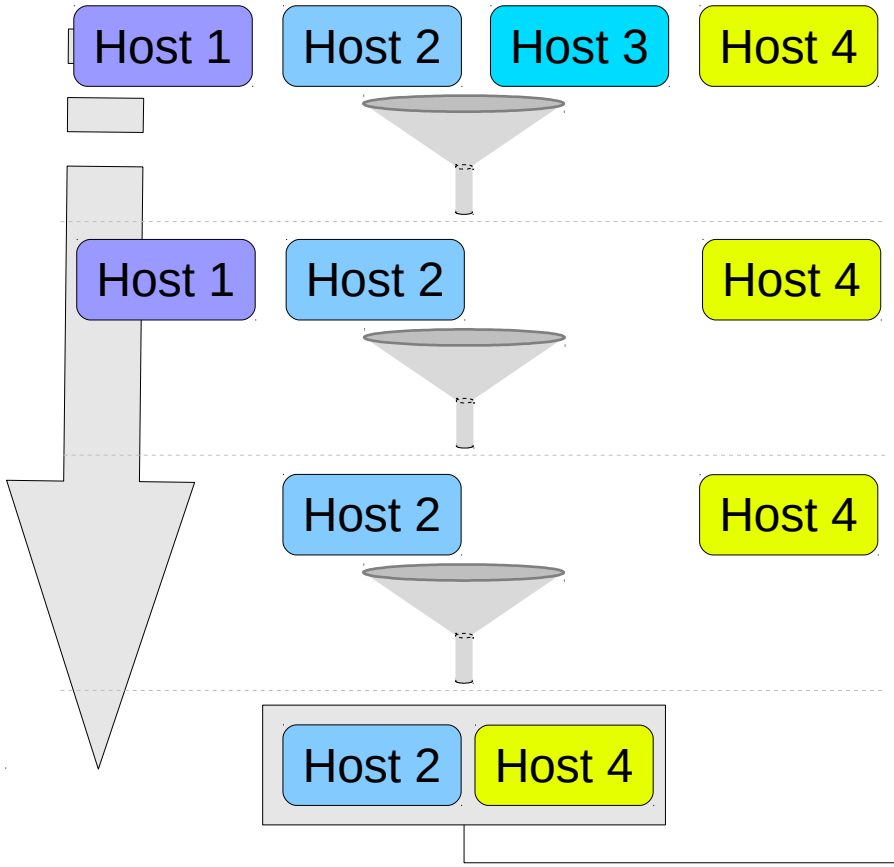
Filter Sample (Java)

```java
@Override
public List<VDS> filter(List<VDS> hosts,
        VM vm, Map<String,
        String> parameters,
        List<String> messages) {

    List<VDS> list = new ArrayList<VDS>();
    for (VDS vds : hosts) {
        Integer cores = SlaValidator.getInstance().getEffectiveCpuCores(vds);
        if (cores != null && vm.getNumOfCpus() > cores) {
            messages.add(VdcBllMessages.ACTION_TYPE_FAILED_VDS_VM_CPUS.toString());
            log.debugFormat("host {0} has less cores ({1}) than vm cores ({2})",
                    vds.getName(),
                    cores,
                    vm.getNumOfCpus());
            continue;
        }
        list.add(vds);
    }

    return list;
}
```

* getEffectiveCpuCores(): checks whether threads count as cores

Weight Module

- The New Model



	func 1	func 2	sum
Factor	5	2	
Host 2	10	2	54
Host 4	3	12	**39***

***Host 4 sum: 3*5+12*2 = 39**

Weight Module

oVirt

- The Weight Module scores each host according to its logic
- Lowest weight is the most preferable candidate
- Weights can be prioritized using Factors; default factor is 1
- Ultimately, we will construct a cost table, which will order the hosts (we will try to run the VM on the best host)

Weight Module (cont.)

- Predefined Weight Modules:
  - Even Distribution
    - Each host weight will be scored according to CPU load, SPMs will be scored higher.
  - Power Saving
    - Define Max_Weight
    - if (no VMs on Host) → Max_Weight
    - Else (Max_Weight – Even_Distribution_Weight)
- External Weight Modules written in python can be loaded into engine.

Weight module Sample

```
class even_vm_distribution():
    '''rank hosts by the number of running vms on them, with the least first'''

    properties_validation = ''

    def do_score(self, hosts_ids, vm_id, args_map):
        #open a connection to the rest api
        try:
            connection = API(url='http://host:port',
                             username='user@domain', password='')
        except BaseException as ex:
            #letting the external proxy know there was an error
            print >> sys.stderr, ex
            return

        #get all the hosts with the given ids
        engine_hosts = \
            connection.hosts.list(
                query=" or ".join(["id=%s" % u for u in hosts_ids]))

        #iterate over them and score them based on the number of vms running
        host_scores = []
        for engine_host in engine_hosts:
            if(engine_host and
                    engine_host.summary):
                host_scores.append((engine_host.id, engine_host.summary.active))
        print host_scores
```

Weight module Sample

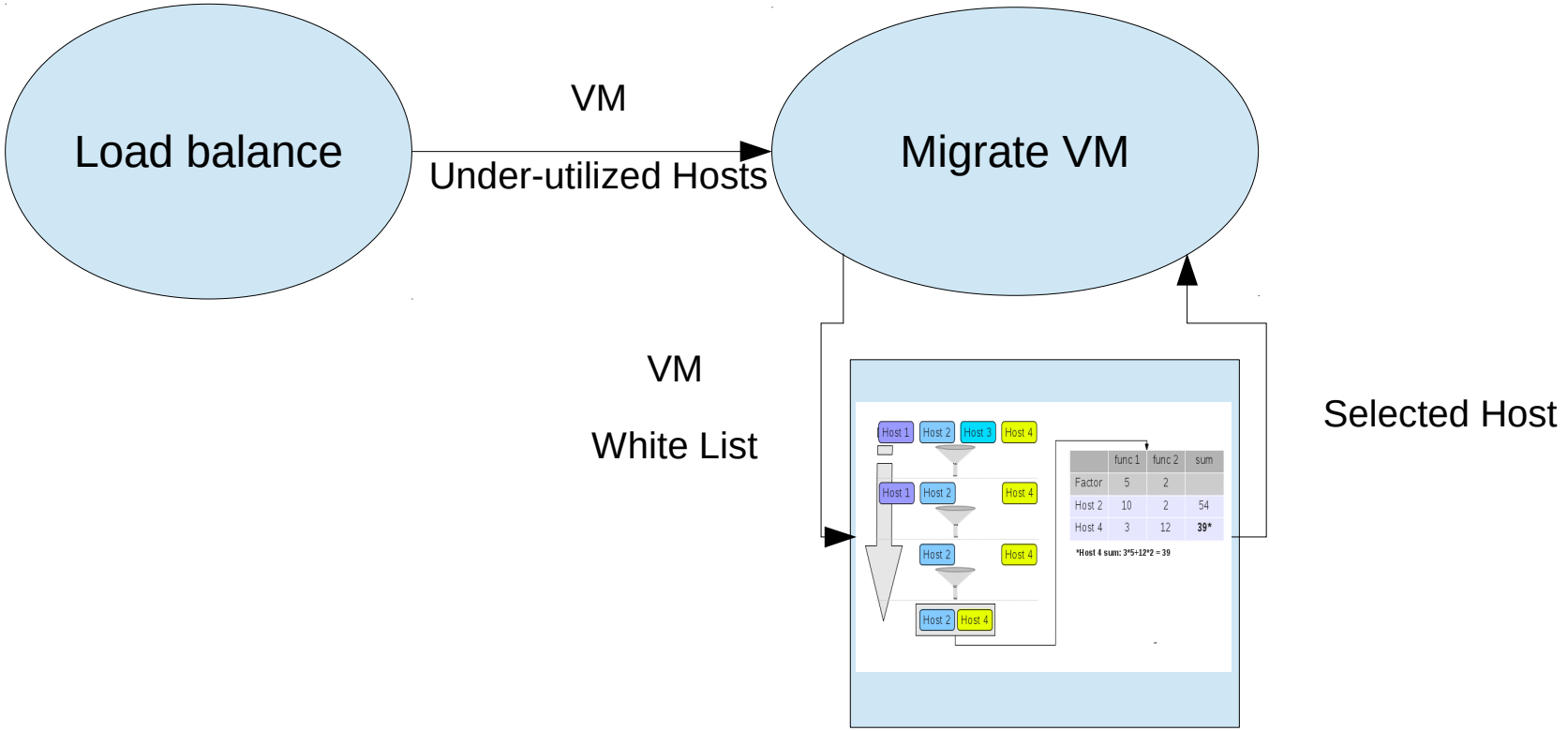
```
class even_vm_distribution():
    '''rank hosts by the number of running vms on them, with the least first'''

    properties_validation = ''

    def do_score(self, hosts_ids, vm_id, args_map):
        #open a connection to the rest api
        try:
            connection = API(url='http://host:port',
                             username='user@domain', password='')
        except BaseException as ex:
            #letting the external proxy know there was an error
            print >> sys.stderr, ex
            return

        #get all the hosts with the given ids
        engine_hosts = \
            connection.hosts.list(
                query=" or ".join(["id=%s" % u for u in hosts_ids]))

        #iterate over them and score them based on the number of vms running
        host_scores = []
        for engine_host in engine_hosts:
            if(engine_host and
                    engine_host.summary):
                host_scores.append((engine_host.id, engine_host.summary.active))
        print host_scores
```

Weight module Sample

```
class even_vm_distribution():
    '''rank hosts by the number of running vms on them, with the least first'''

    properties_validation = ''

    def do_score(self, hosts_ids, vm_id, args_map):
        #open a connection to the rest api
        try:
            connection = API(url='http://host:port',
                             username='user@domain', password='')
        except BaseException as ex:
            #letting the external proxy know there was an error
            print >> sys.stderr, ex
            return

        #get all the hosts with the given ids
        engine_hosts = \
            connection.hosts.list(
                query=" or ".join(["id=%s" % u for u in hosts_ids]))

        #iterate over them and score them based on the number of vms running
        host_scores = []
        for engine_host in engine_hosts:
            if(engine_host and
                    engine_host.summary):
                host_scores.append((engine_host.id, engine_host.summary.active))
        print host_scores
```

Load Balancing

- Triggers a scheduled task to determine which VM needs to be migrated to one of under-utilized hosts
- A single load balancing logic is allowed per cluster



Load Balancing (cont.)

oVirt

- For backward compatibility we have 2 predefined Load Balancing algorithms
  - Even Distribution:
    - Calculates over-utilized and under-utilized hosts according to upper CPU load threshold
    - Select a VM out of the over-utilized hosts.
    - Pass VM and under-utilized hosts to the scheduler
    - migrate VM to the host selected by the scheduler
  - Power Saving:
    - Same as Even Distribution, but with a second threshold for low CPU load
- External load balancing written in python can be loaded into engine

Load Balance (Sample)

… same as previous

```
#iterate over them and decide which to balance from
over_loaded_host = None
white_listed_hosts = []
for engine_host in engine_hosts:
    if(engine_host):
        if (engine_host.summary.active < maximum_vm_count):
            white_listed_hosts.append(engine_host.id)
            continue
        if(not over_loaded_host or
                over_loaded_host.summary.active
                < engine_host.summary.active):
            over_loaded_host = engine_host

if(not over_loaded_host):
    return

selected_vm = None
#just pick the first we find
host_vms = connection.vms.list('host='+over_loaded_host.name)
if host_vms:
    selected_vm = host_vms[0].id
else:
    return

print (selected_vm, white_listed_hosts)
```

Load Balance (Sample)

… same as previous

```
#iterate over them and decide which to balance from
over_loaded_host = None
white_listed_hosts = []
for engine_host in engine_hosts:
    if(engine_host):
        if (engine_host.summary.active < maximum_vm_count):
            white_listed_hosts.append(engine_host.id)
            continue
        if(not over_loaded_host or
                over_loaded_host.summary.active
                < engine_host.summary.active):
            over_loaded_host = engine_host

if(not over_loaded_host):
    return

selected_vm = None
#just pick the first we find
host_vms = connection.vms.list('host='+over_loaded_host.name)
if host_vms:
    selected_vm = host_vms[0].id
else:
    return

print (selected_vm, white_listed_hosts)
```

Load Balance (Sample)

… same as previous

```
#iterate over them and decide which to balance from
over_loaded_host = None
white_listed_hosts = []
for engine_host in engine_hosts:
    if(engine_host):
        if (engine_host.summary.active < maximum_vm_count):
            white_listed_hosts.append(engine_host.id)
            continue
        if(not over_loaded_host or
                over_loaded_host.summary.active
                < engine_host.summary.active):
            over_loaded_host = engine_host

if(not over_loaded_host):
    return

selected_vm = None
#just pick the first we find
host_vms = connection.vms.list('host='+over_loaded_host.name)
if host_vms:
    selected_vm = host_vms[0].id
else:
    return

print (selected_vm, white_listed_hosts)
```

oVirt

Q&A
for 1st part*?

* Coming up: internal implementation...

Engine Implementation: Policy Unit

oVirt

- New Entity
- Basic logical building block for scheduling, a set of policy units construct a cluster policy
- Holds meta-data for a single policy logic:
  - Each logic can represent Filter or Weight or balancing module
- Internal policy units (pin to host, memory, etc.) are predefined

Policy Unit (cont.)

oVirt

- Structure
  - name (based on class name)
  - description (used for tool-tips)
  - type: filter/weight module/load balancing
  - is_internal: represents either internal or external units.
  - Allow custom properties per unit (key1=regex1;key2=regex2)
  - In case there is inconsistency between external units to DB stored (and used) units, the unit is marked as disabled.

Cluster Policy

oVirt

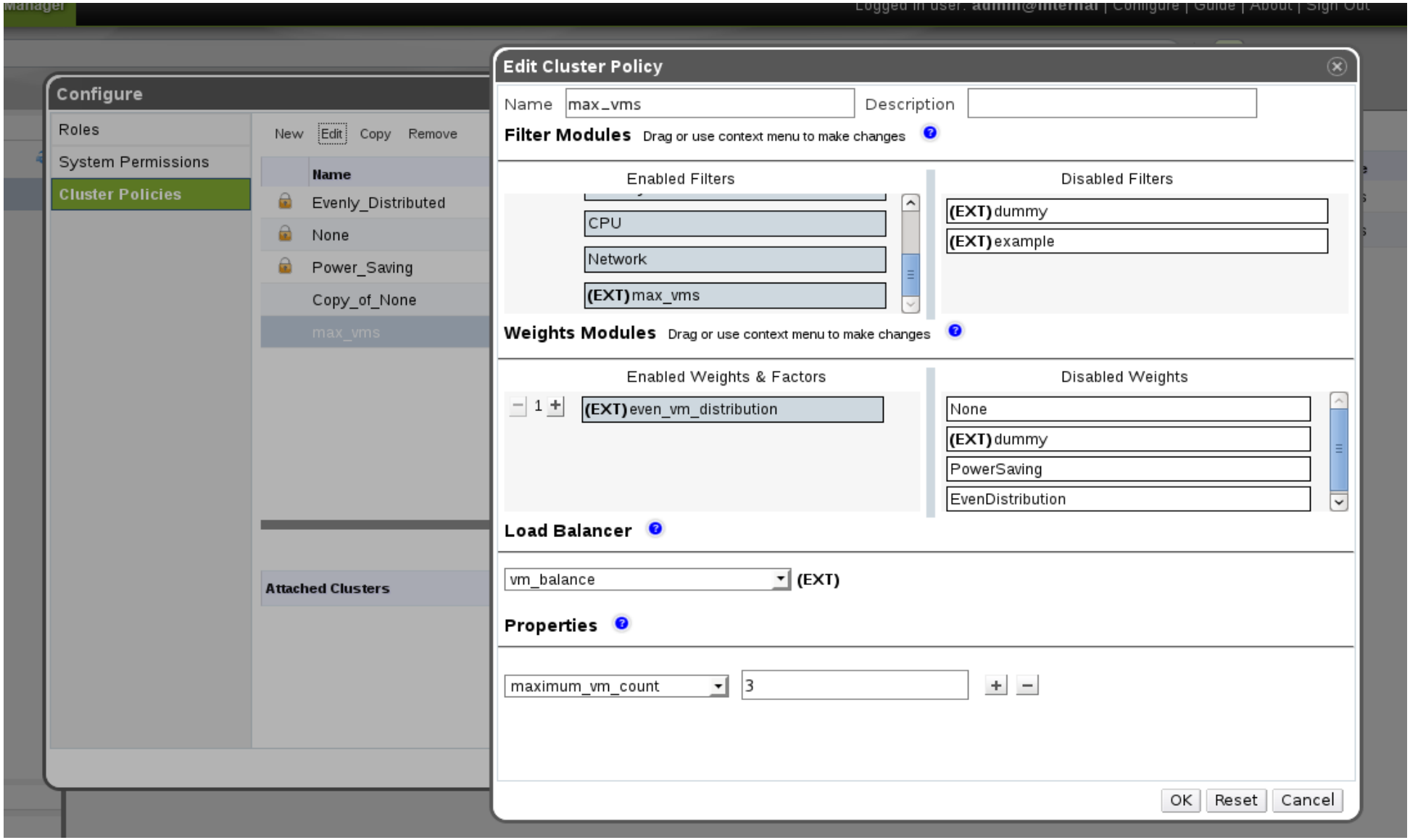
- New Entity
- Holds a collection of policy units to form a cluster's scheduling policy
- Each cluster policy can be attached to multiple clusters, and its custom parameters can be overridden
- Former policies (None, Evenly Distribution and Power Saving) are migrated to the new arch as predefined cluster policies

Cluster Policy (cont.)

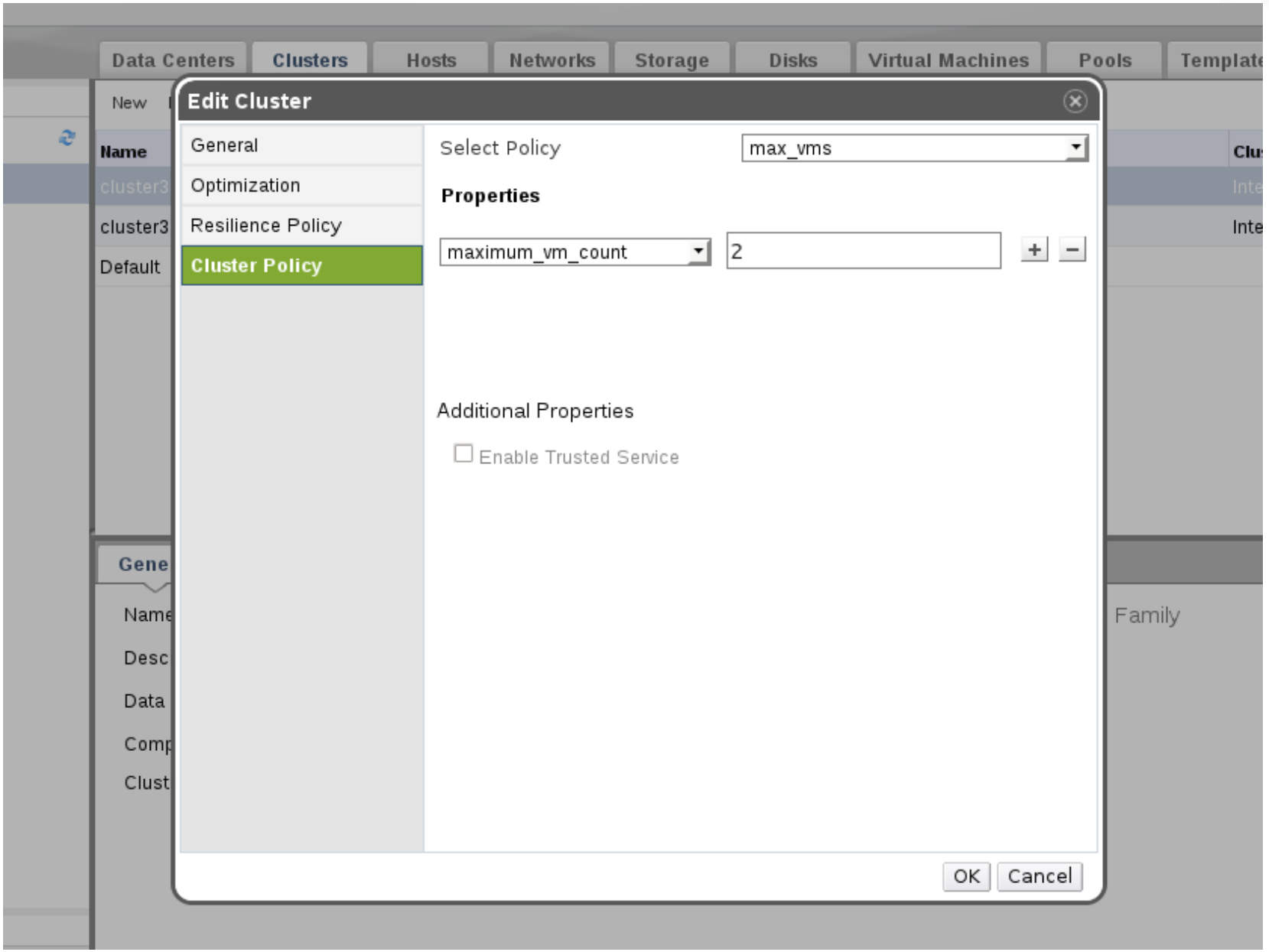
oVirt

- Structure
  - name
  - description
  - list of filters
    - Execution order is insignificant
    - Optional: Filter position (one filter may be set to run first, and one last)
  - list of weight modules and factors
  - single load balancing logic
  - allows to set custom properties per policy according to policy units.
  - is_locked

Cluster Policy Management



Attach Cluster Policy



Scheduling Manager

oVirt

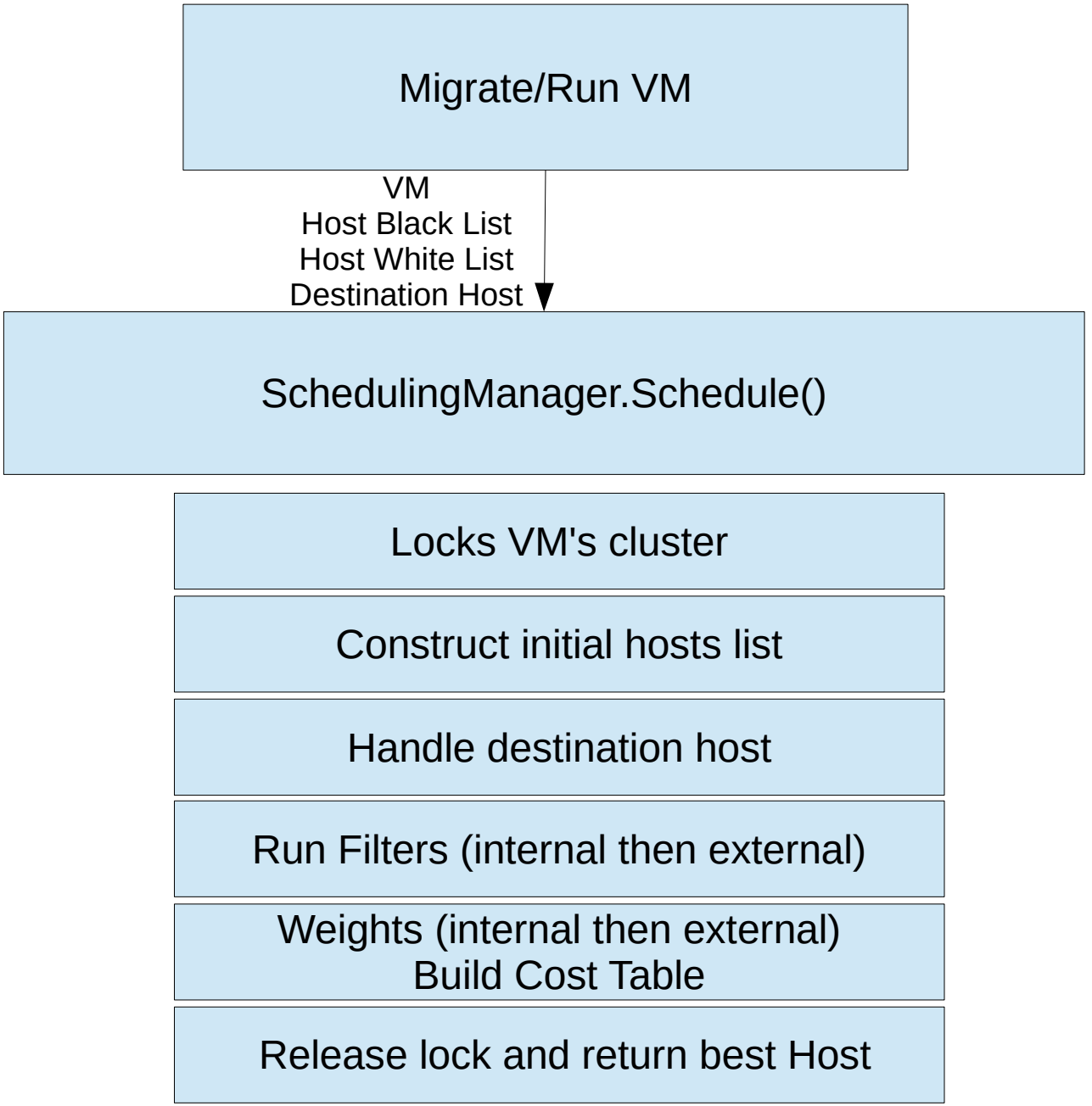
- We define a new singleton object, SchedulingManager
- Responsible for all scheduling activities
- Initialize scheduled Load Balancing Task
    - According to engine configuration (enabled, interval)
- Serves run/migrate VM scheduling requests
- Loads and holds policy units and cluster policies
- Interacts with external scheduler proxy

Scheduling Manager (cont.)

oVirt

- Loads Policy Units & cluster policies
  - Loads from DB all stored entities to memory maps.
  - External (if needed):
    - Run Discover command (in a separate thread) to fetch all available external policy units.
    - Compares loaded policy units with discovered ones.
      - Missing modules mark as disabled.
      - New are added to DB.
      - Modified are updated in DB.
    - Refresh policy units in memory cache.

Flow: Schedule Request



DB Upgrade

oVirt

- Insert predefined policy units
- Insert Predefined Cluster Policies for Even Distribution, Power Saving and No Balancing
  - Each predefined cluster policy is made of internal policy units
- Each cluster will point to a cluster policy according to its selection algorithm.
- Other selection algorithm parameters will be migrated to a properties map.

External Scheduler

oVirt

- External service written in python and run as a separate process from the engine
- Why do we need it?
  - Engine safety
  - Should allow other languages
  - Going forward we may suggest SaaS (Scheduling as a Service)

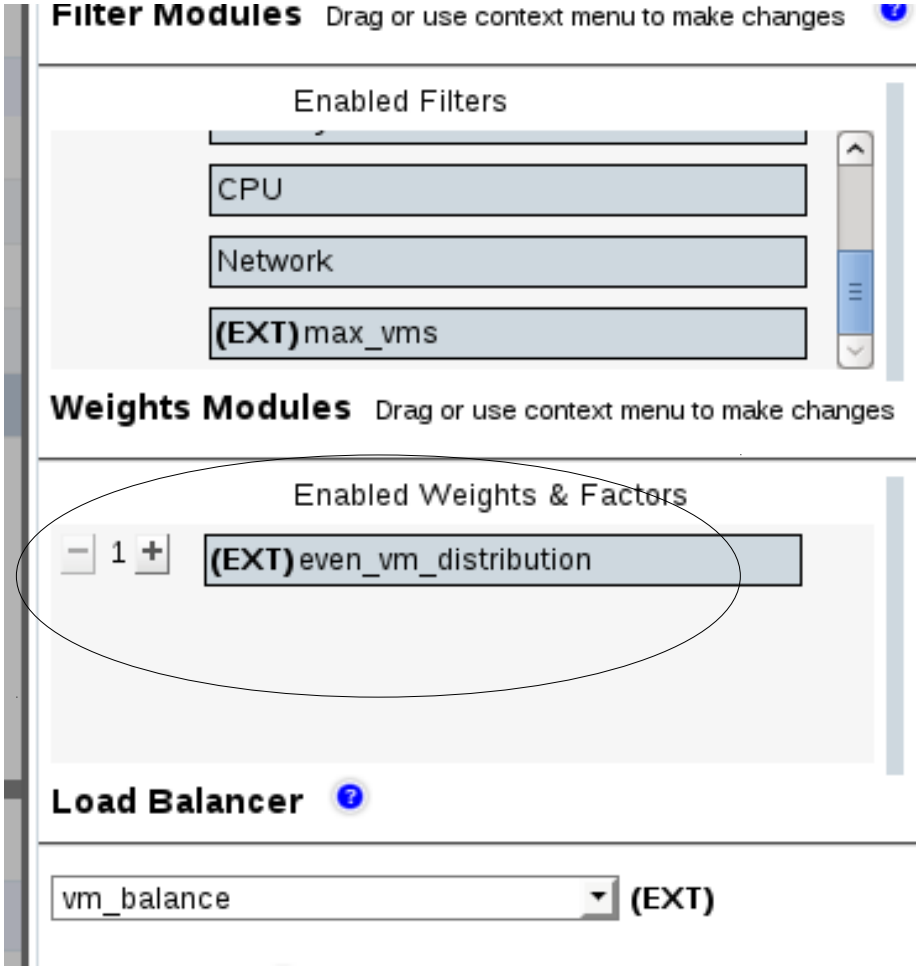
External Scheduler (cont.)

oVirt

- Packaged as ovirt-scheduler-proxy RPM, which is optional (not installed by default).
- Initialization
  - Service Start
  - Analyze
  - Publishing Internal API (Starting XML-RPC Server)
  - Waiting for engine calls
    - Discover
    - ...

External Code Representation

- Init:
  - Scan /usr/share/ovirt-scheduler-proxy/plugins for *.py
    - Analyze for filters/weights/balance
    - Cache results
- Discover: return cached results



External Scheduler (cont.)

oVirt

- RunFilters (or Weights/Balance)
  - Filters names
  - UUIDs as parameters, args_map
- Start process for each Filter
  - Pass parameters in process initialization
  - Wait (with timeout) for process
  - Communicate using stdout/stderr to get results
- Aggregate results for all processes
- Return result to engine

Future

oVirt

- Schedule multiple VMs
- Loadable Java plug-ins
- SaaS: Scheduling as a service, which will allow us several scheduling services

oVirt

Q&A

Thank you :-)